NASA/TM—2002-211358 AIAA–2002–0750

Project Integration Architecture (PIA) and Computational Analysis Programming Interface (CAPRI) for Accessing Geometry Data From CAD Files

Theresa L. Benyo
Glenn Research Center, Cleveland, Ohio

March 2002

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the Lead Center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized data bases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA Access Help Desk at 301–621–0134

- Telephone the NASA Access Help Desk at 301–621–0390

- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076

NASA/TM—2002-211358

AIAA–2002–0750

Project Integration Architecture (PIA) and Computational Analysis Programming Interface (CAPRI) for Accessing Geometry Data From CAD Files

Theresa L. Benyo
Glenn Research Center, Cleveland, Ohio

Prepared for the
40th Aerospace Sciences Meeting and Exhibit
sponsored by the American Institute of Aeronautics and Astronautics
Reno, Nevada, January 14–17, 2002

National Aeronautics and
Space Administration

Glenn Research Center

March 2002

Available from

NASA Center for Aerospace Information
7121 Standard Drive
Hanover, MD 21076

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22100

Available electronically at http://gltrs.grc.nasa.gov/GLTRS

PROJECT INTEGRATION ARCHITECTURE (PIA) AND COMPUTATIONAL ANALYSIS PROGRAMMING INTERFACE (CAPRI) FOR ACCESSING GEOMETRY DATA FROM CAD FILES

Theresa L. Benyo*
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

ABSTRACT

Integration of a supersonic inlet simulation with a computer aided design (CAD) system is demonstrated. The integration is performed using the Project Integration Architecture (PIA). PIA provides a common environment for wrapping many types of applications. Accessing geometry data from CAD files is accomplished by incorporating appropriate function calls from the Computational Analysis Programming Interface (CAPRI). CAPRI is a CAD vendor neutral programming interface that aids in acquiring geometry data directly from CAD files. The benefits of wrapping a supersonic inlet simulation into PIA using CAPRI are; direct access of geometry data, accurate capture of geometry data, automatic conversion of data units, CAD vendor neutral operation, and on-line interactive history capture. This paper describes the PIA and the CAPRI wrapper, and details the supersonic inlet simulation demonstration.

INTRODUCTION

Acquiring geometry input data for computational fluid dynamics (CFD) simulations consumes valuable time and often results in incomplete or inaccurate data. The Project Integration Architecture (PIA) along with the Computational Analysis Programming Interface (CAPRI) provides an environment for accessing geometry directly from CAD files and making the data available to CFD simulations. The PIA is an object-oriented, wrapping architecture for capturing, encapsulating, presenting, and integrating all elements of day-to-day technical aerospace research activity.

The benefits of PIA are:

- Direct access to data of many formats
- Accurate capture and presentation of information
- Convenient data archiving in a single environment

CAPRI is a programming interface for acquiring geometry data directly from CAD files in a vendor-neutral manner. The wrapping of CAD information by PIA through the use of CAPRI provides geometry objects that hold and organize the data.

PIA[1] provides a common, self-revealing application architecture that eliminates the need to repeatedly adapt graphical user interfaces (GUIs), browsers, search engines, and other applications to various experimental and analytical information sources. This architecture uses object-oriented technology to implement application wrappers that encapsulate, present, and integrate all elements of day-to-day technical information. This information includes data pertaining to experiments, designs, analyses, and simulations. Further, this information can form the foundation upon which statistical characterizations and optimizations are based. The self-revealing architecture of PIA allows consumers of application information to use a particular application without pre-existing knowledge of the application contents.

An application presented through a PIA-conformant wrapper begins with a central application object, labeled **PacAppl** in the upper center of Figure 1. This object is the root structure from which all further components emanate. The **PacAppl** object currently presents four principal structures:

* Computer Engineer, AIAA Member. E-mail: Theresa.Benyo@grc.nasa.gov

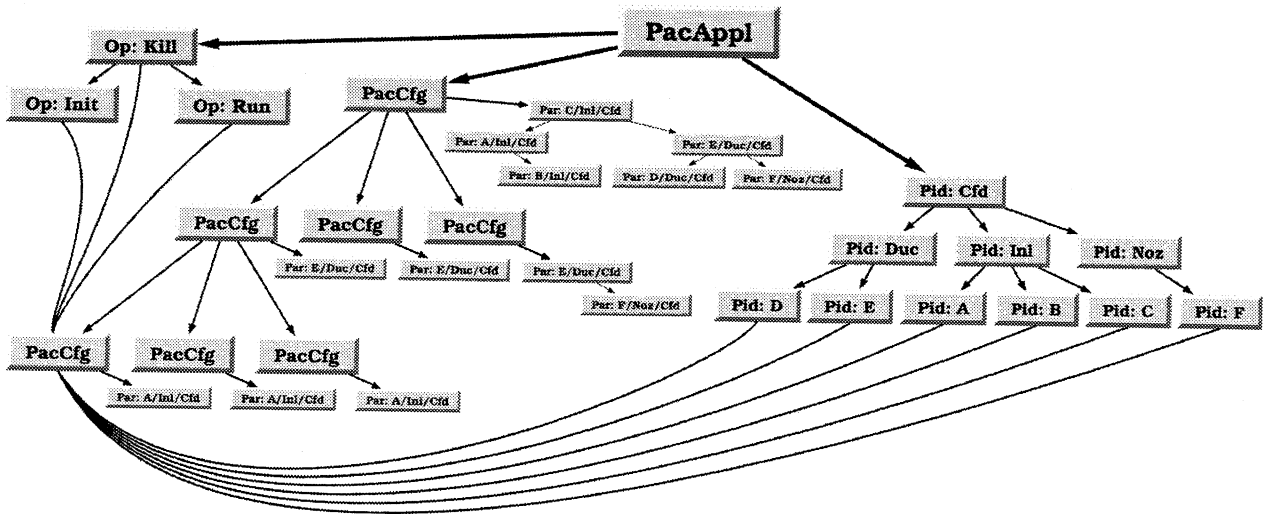

Figure 1: The PIA object classes

1. A set of operations that the application is willing to perform (**PacOp**),
2. A mass of data, which the application currently contains (**PacCfg**),
3. A structure by which the contained data is identified (**PacPid**), and
4. An ecdysiastical (from the Greek *ekdysis, ekdyein,* to get out of, strip off) sorting of the information-bearing objects in the application (not shown in the figure).

The **PacOp** structure, as illustrated in Figure 1, could contain operations such as Initialize, Run, and Stop. The implementer of the particular application wrapper has complete freedom to attach any kind or number of operations needed. As will be discussed later, the PIA/CAD/CAPRI wrapper implements only an **AcquireData** operation.

The **PacCfg** structure organizes the parameter objects (the blocks beginning with the label **Par:** in the figure) that hold the data of the application. For a computational fluid dynamics (CFD) application, these objects could hold boundary layer information, grid coordinates, or geometry describing the modeled environment. Parameters within a given **PacCfg** are sorted by a unique, fully qualified name, which is to be discussed shortly. The **PacCfg** objects themselves are arranged into an *n*-ary tree in which offspring are considered to be variants of their ancestors. A parameter missing in a descendant configuration is considered to be inherited from the most recent ancestor containing that parameter. In this way, needless repetition of invariant information is avoided.

The parameter objects as mentioned above, hold the actual data of applications. A wide variety of parameter forms; Booleans, strings, integers, floating point numbers, scalars, arrays, matrices, organizations of other parameters, and so on, are defined. After the generic forms are defined, the semantics of a particular form are infused by means of further class derivation. Of most interest in this way are the floating-point number forms, which form the basis of a vast array of technical parameters. First, the floating point numbers are infused with a concept of their own dimensionality, both in terms of their dimensional characteristics (length, mass, velocity, non-dimensional, etc.) and of the measurement system in which their encapsulated value is given (English feet, English inches, metric meters, metric centimeters, etc.). This allows applications to proceed without concern for the measurement systems in which they operate; values are simply requested in the desired measurement system (a CFD application may work in feet, pounds, and seconds) and the parameters convert themselves as necessary. After this, further class derivation gives the number a usage, for example a non-dimensional floating-point number is further derived into a Mach number and then to a far-field Mach number. An application encountering such a derived object thus has the capability of determining what it is (a far-field Mach number) and deciding whether or not that is the sort of information it wishes to find.

The **PacPid** structure exists to reveal the structure of parameters within the application. Again, the structure is arranged into an *n*-ary tree in which offspring are considered grouped under the parent. The fully qualified name of a parameter (used to identify the parameter in the **PacCfg** configuration discussed above) is developed by concatenating the names of the corresponding path in the **PacPid** tree. In the example shown in Figure 1, the root of the identification tree is **Cfd**, which in turn has three direct offspring, **Duc**, **Inl**, and **Noz** (presumably, an inlet, duct, and nozzle). **Duc**, in turn, has two direct offspring, **D** and **E**. The fully-qualified name of the D parameter would then be **D/Duc/Cfd**, as it appears in the **PacCfg** portion of the figure.

The fourth component, the ecdysiastical sorting (which is not shown in the figure), serves to provide quick access for entities such as browsers and search engines to well-known types of information within the particular application wrapper instance, even though that information may not exist exactly in its well-known from. PIA allows application wrappers to employ the derivative capacities of object-oriented technology to specialize parameters beyond their well-known character (as is, in fact, the case in the PIA/CAD/ CAPRI/ProEngineer wrapper). As a consequence, a parameter may not be well-known on its face, but through the ecdysiastical sorting, it still may be quickly located based on its underlying character.

Together, this application structure enables researchers to maintain and manage experimental data, simulations, analyses, documentation, logs, change histories, and many other forms of information in a common repository that is easily accessed and extended. As a result, the entire engineering process can be captured. The well-known nature of the many objects of which this architecture is comprised enables the integration of these many technical components into a logical whole.

PIA enables a common user interface and allows browsers and search engines to deal with the myriad of technical information applications in a common manner. PIA also eliminates the numerous manual steps in exchanging data between different disciplines and levels of fidelity, resulting in a framework for the automation of routine tasks.

CAPRI

In order to achieve the goal of gathering geometry data from Computer Aided Design (CAD) files into the PIA environment, it is necessary to wrap this application in a PIA-compliant wrapper. The technology provided by CAPRI[2] to provide a vendor-neutral interface to this information was utilized to avoid having to provide a wrapper specific to each CAD vendor.

CAPRI provides a library specific to each CAD vendor. Each library implements the common CAPRI Application Programming Interface (API) using services specific to the supported vendor. By programming to the CAPRI API and linking to the appropriate vendor-specific library, a consuming application may be made independent of the particular CAD vendor from which geometry information is to be obtained. Currently, CAPRI provides libraries to support Unigraphics, ProEngineer, CATIA, FELISA, Computervision's CADDS, and SDRC's I-DEAS products.

CAPRI provides geometry information in a data hierarchy of nodes, edges, faces, boundaries, and volumes. Figure 2 illustrates this hierarchy. Nodes are the simplest entities and are just points in 3-space. Edges are open curves. Edges begin and end at distinct nodes and, thus, a closed curve must be formed by two or more edges. Faces are bounded by closed sets of edges organized into loops and may join other faces at shared edges. Boundaries then collect faces together into sets. Volumes are closed regions of 3-space bounded by the sum of all the faces found in the boundaries of the volume.

CAPRI also provides tessellations of edges and faces. Edges are tessellated as an ordered stream of points in 3-space. Faces are tessellated as points in 3-space arranged into triangles. Information on the connectivity of the triangles within a face is provided. The points used to tessellate an edge are identically those used to tessellate the edge of the faces, which that edge terminates so that a complete triangulation of the volume as a whole is obtained.

An additional object called a bounding box is also provided. Bounding boxes are items that indicate the 3-space that a particular CAPRI data object such as a face is in.

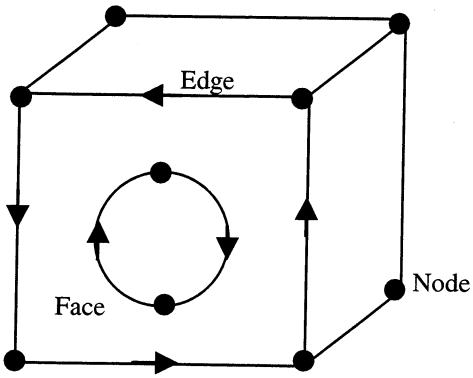

Figure 2: A simple volume with a cylinder cutout—Edges marked with arrows for front face

PIA has defined well-known parameter objects that follow the structuralization of geometry information provided by the CAPRI technology. The PIA/CAD/CAPRI/ProEngineer wrapper uses the services of CAPRI to obtain geometry information from a ProEngineer CAD file, create and populate corresponding PIA parameter objects, and place those parameter objects in a parameter configuration, and create the corresponding parameter identification structure. The **PacAppl**-based object of this wrapper contains a module, which reads an identified CAD file through CAPRI facilities, interprets the data found, and performs all the appropriate object creation and organization.

PIA/CAD/CAPRI WRAPPER IMPLEMENTATION

This section describes the implementation of the PIA/CAD/CAPRI wrapper that captures and presents geometry information from CAD files using the CAPRI technology. Figure 3 shows three of the four application structures defined by the PIA; the operations available (in this case, only the **AcquireData** operation), the single parameter configuration created, and the parameter identification structure.

The root of the application, **CpeAppl** (Capri-Pro/Engineer Application) is a derivative of the generic application class, **PacAppl**, defined by the PIA architecture. This derivative class provides the specifics to convert the generic application shell into a real application wrapper, in this case of CAD data obtained from ProEngineer through CAPRI technology. One significant and well-known function, **CreateApplication**, must be overridden by this derivative class. In this case, this function acquires geometry information and creates the well-known geometric parameter objects that are the ultimate goal of the implementation. Additionally, a number of operations and facilities are implemented in this derivative application class that are not well-known, but cooperate to implement and achieve the well-known

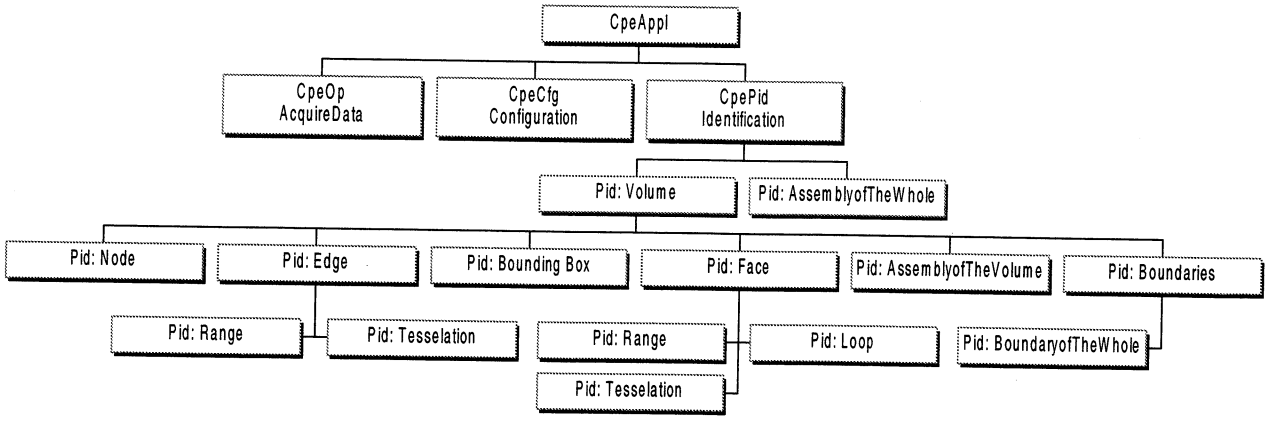

Figure 3: Architecture of the PIA/CAD/CAPRI application wrapper

result of a PIA-compliant application wrapper. Included in this application-specific area is the ability to communicate with a backend, geometry server that is necessitated by the exigencies of CAPRI/ProEngineer operation.

The single operation provided, **AcquireData**, prompts the user through PIA-defined facilities, to identify a CAD file from which geometry information is to be obtained. Once this file is selected, the operation starts the backend, geometry server mentioned above, transmits the file selection to it, and receives from the backend server the object-encapsulated geometry data and identification information acquired from that file.

The parameter objects acquired by the **AcquireData** operation are placed in the single parameter configuration object implemented by the wrapper. While the PIA defines a configuration hierarchy with descendent parameter configurations, parameter inheritance, and so on, this concept does not presently exist within the CAPRI technology; within CAPRI, there is only the geometry data. Thus, while descendent parameter configurations may be created within a PIA/CAD/CAPRI wrapper, no provision presently exists for populating them with any additional geometric parameter objects and all geometric information in such a descendent configuration will be, in fact, inherited from the root of the parameter configuration tree.

The parameter-identification structure, illustrated in Figure 3, is built by the **AcquireData** operation based upon the information it receives from the backend, geometric server. The structuralization of geometric parameters closely follows that defined by CAPRI with volumes containing nodes, edges, faces, boundaries, and a bounding box, edges containing parameterization ranges and edge tessellations, and faces containing parameterization ranges, loops, and face tessellations. Three additional parameters called AssemblyofTheWhole. AssemblyofTheVolume, and BoundaryofTheWhole are created in the wrapper. AssemblyofTheWhole creates an application-wide assembly of every boundary that the wrapper encounters. As a result, the whole geometry can be visualized. AssemblyofTheVolume creates an assembly for a volume if there is more than one boundary. BoundaryofTheWhole creates a boundary if there are no boundaries provided by CAPRI and groups all the faces into one. Names for the various geometric components are numerically based. For example, the fourth face of a given boundary becomes, simply, Face4.

Figure 4 shows how the concatenation of names from the identification tree is used to identify particular parameters in the parameter configuration. For example, the first loop of the second face of the first volume would be Loop1/Face2/Volume1, as shown in the figure. The figure only shows a partial representation of the parameter configuration.

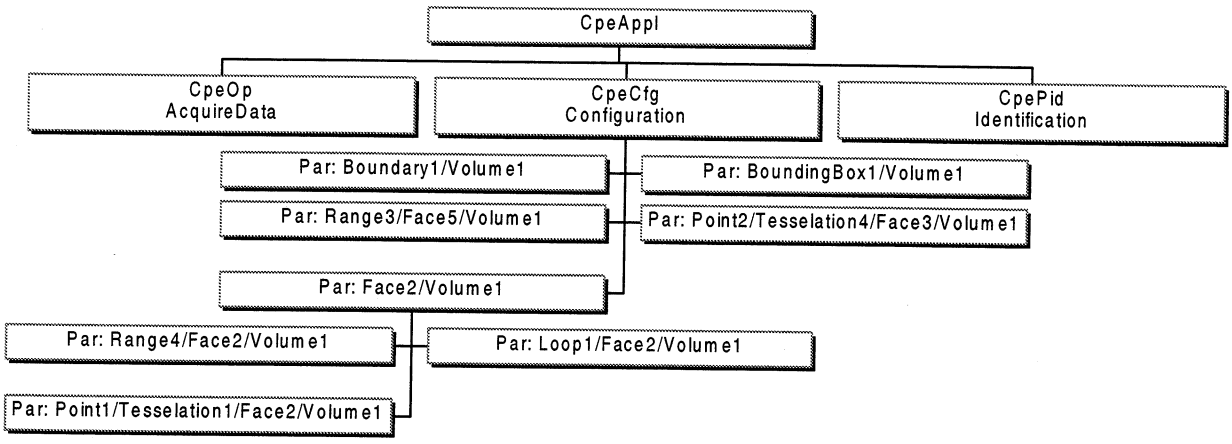

Figure 4: Example configuration of PIA/CAD/CAPRI application wrapper

IMPLEMENTATION OF THE BACKEND, GEOMETRY SERVER

Acquisition of geometry information is, for the user, quite simple: the appropriate CAD file is identified and magic happens. The efforts to which the wrapper goes to make this magic happen are somewhat more extensive.

While CAPRI has achieved apparent vendor-neutrality at the API level, this neutrality and consequent ease does not extend to the actual making of an executable program. One does not merely link the correct CAPRI library into a main program and go from there. The mechanisms necessary to obtain a working program from the selection of the correct CAPRI library can vary and the results are, at times, not convenient. Much, if not all, of this is attributable not to some failure of the CAPRI research effort, but simply to the different forms and modalities in which the various CAD vendor products offer access to the raw geometric information with which CAPRI works.

In the case of ProEngineer, an optional software component, ProToolkit, executes the vendor's geometry kernel and then links to and executes a dynamic link library identified to it when ProToolkit is started. The CAPRI library, and the "main program" invoking it are, in fact, subprograms of the ProToolkit execution, which is, itself, a spawned process of a batch file, which establishes the appropriate environment for its operation. Since the graphical user interface through which the PIA is exercised regards itself as a patriarchal process, this process structure of ProToolkit and its CAPRI access library presents a certain difficulty that is dealt with as described in the following paragraphs.

The **CpeAppl** specialization of the application object implements a client characteristic which, when TRUE, indicates that it is operating as the apparent, PIA-compliant, frontend application wrapper. When **AcquireData** operates, it locates its containing **CpeAppl** application object and interrogates it for the state of this characteristic. Finding itself to be a part of a client, **AcquireData** starts up the batch file (in which it has placed the name of the desired CAD file, as well as some socket communication information) to start the ProToolkit execution. When ProToolkit connects to the identified dynamic link library and calls its entry point, the "main program" does the following. It receives the transmitted file and communication information. It creates another **CpeAppl** object and informs that object that it is not a client (that is, the client characteristic is made FALSE). It calls the **CpeAppl** object's **CreateApplication** member function (which extracts through CAPRI all the geometric information and encapsulates it in objects), and then informs the frontend client that the backend, geometric server is ready for operations. When **AcquireData** receives this ready signal, it sends a message requesting the transmission of the object-encapsulated, geometric data, which it then receives and places in the structures of its containing **CpeAppl** client object.

The operation of the **CreateApplication** member function, while long and tedious, is not particularly complicated. CAPRI functions are called to obtain geometric data and create objects to encapsulate and identify it as it is found to exist. For example, one CAPRI function is called to determine how many volumes exist. A loop is then executed to create a volume identification structure and obtain the specific information for each volume in turn. Each volume indicates how many nodes, edges, faces, and boundaries exist in it and internal loops are executed to identify and obtain information for each of these in their turn. Because of the close correlation between the geometric structuralizations used by CAPRI and PIA, this process is very natural and relatively easy to implement.

WRAPPER-SPECIFIC PARAMETER CLASSES

Some of the geometric parameter classes defined by PIA provide geometric services beyond that of simple data presentation. For example, the boundary class provides the ability to obtain a cross section of its geometric shape. The implementation of this function provided by the well-known PIA parameter class is based solely upon the information contained in the face tessellations associated with the faces of the boundary. Unfortunately, such cross sections, when based simply upon triangular tessellations introduce noise into the geometric information when the tessellated face exhibits some finite curvature. The sides of the triangles represent chords relative to the face curvatures they attempt to describe and, thus, some deviation between the practical and the ideal exists.

CAPRI offers a potential cure to this difficulty in the form of a snap-to-face functionality. The cross-sectional position computed from the tessellating triangle's side may then be improved by snapping it back onto the geometric face, thus reducing introduced error to some acceptable value. Unfortunately, this snap-to-face functionality is only available with a live, operating CAPRI and is, thus, unavailable to the **CpeAppl** client frontend wrapper.

To alleviate this difficulty, the PIA/CAD/CAPRI wrapper derives several of these geometric parameter objects beyond their well-known level and adjusts the functionality in these particular cases. Continuing the example above, the boundary parameter object now knows that it might be a member of a client **CpeAppl** wrapper. It locates its containing **CpeAppl** application and determines if this is the case. In this event, it does not do the cross-sectioning operation itself (which it inherits from its well-known base class), but instead transmits a message (through **CpeAppl**-specific facilities) to its counterpart in the backend server. That counterpart first invokes the inherited functionality to obtain a basic cross-section result and then utilizes the snap-to-face functionality provided by CAPRI to improve that result and, to a specified level, remove the induced geometric noise. The final result is then transmitted back to the frontend client that returns it to its caller as its own work.

RBCC HYPERSONIC VEHICLE EXAMPLE

The acquisition and presentation of geometric data through the PIA/CAD/CAPRI wrapper in the manner described above has been demonstrated with the geometry of a Rocket-Based Combined Cycle (RBCC) hypersonic vehicle propulsion system under study at the Glenn Research Center. This information was then examined by a PIA wrapper of the Large Perturbation Inlet (LAPIN) simulation code. That second wrapper used the facilities of the presented geometric parameters (in particular, the cross-sectioning facilities) and its own heuristics to generate the LAPIN-specific flowpath information needed for LAPIN operation. LAPIN was then executed and its results encapsulated in parameter objects presented by that second, PIA/LAPIN wrapper. All of this proceeded on an automated basis.

Figure 5: RBCC propulsion system as displayed by CAPRI

Figure 5 shows a CAPRI/ProEngineer rendering of the RBCC propulsion system assembly. Figure 6 shows a rendering by the research graphics user interface (GUI) of the geometric information of the RBCC propulsion system as presented by the PIA/CAD/CAPRI wrapper. (Note that the research GUI to PIA is a research tool only and not a project product; thus, less than completely sophisticated renderings and displays are considered entirely adequate performance for the GUI component.) Also illustrated toward the upper left corner of Figure 6 is a partially expanded portion of the identification structure resulting from the geometric information obtained by the PIA/CAD/CAPRI wrapper through the CAPRI interface to the original ProEngineer data.

Using the implemented PIA/CAD/CAPRI/ProEngineer wrapper, the RBCC propulsion system information contained in the original ProEngineer CAD file can be read and the data encapsulated in geometric objects as described above. Once this geometric information is presented in this manner, another consumer of information (in the case of the present demonstration, the PIA/LAPIN wrapper) can access and consume this data as it deems appropriate. Note that the information, by the design of PIA, is accessed by its kind, that is as being a geometric boundary or a geometric face, and not by the application presenting it. Thus, the PIA/LAPIN wrapper is not dependent upon finding a PIA/CAD/CAPRI wrapper for its information, but only upon finding some wrapper containing geometric boundary information.

The PIA/LAPIN wrapper is programmed, having found a source of geometric-boundary information, to convert that information into the unique flowpath description form required for LAPIN operation. LAPIN-specific processes and rules are encapsulated in the PIA/LAPIN wrapper to effect this result. These processes utilize parameter-provided services, in particular the ability to provide cross sections of geometric boundaries and the ability to manipulate, sort, and characterize the curves

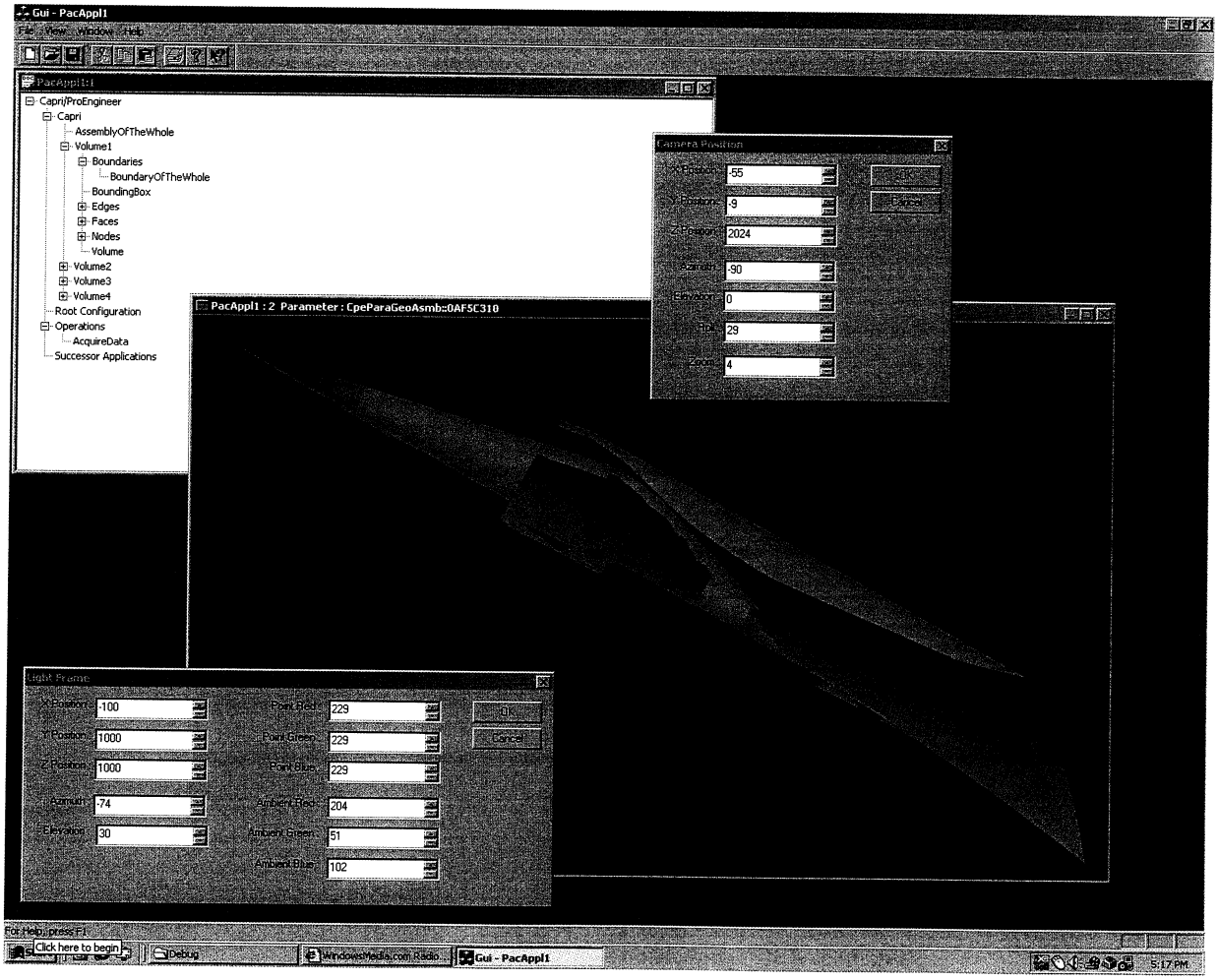

Figure 6: RBCC propulsion system information in PIA

that result. Processing and rules that are LAPIN-specific (i.e., the discrimination between an inlet with no centerbody and a two-dimensional inlet with no second cowl cross-section) are encapsulated in the interior of the PIA/LAPIN wrapper and are not presented to the outside PIA environment.

A complication presented in the RBCC propulsion system example is that the flowpath of this system is neither axisymmetric nor two-dimensional. Instead, the flowpath has a cross section akin to the shape of an orange segment extending around one third of the circumference of a round fuselage. Since LAPIN works internally in terms of cross-sectional area, presenting the simple XY cross section of the flowpath to LAPIN produced erroneous analyses under LAPIN's default axisymmetric assumption. Similarly, adjusting either XY profile to produce appropriate cross-sectional areas led to incorrect calculation of oblique shock points. The accepted solution was to describe the inlet under LAPIN's two-dimensional geometry option with width factors adjusted to produce correct cross-sectional areas. The difficulty with both these later approaches was that the development of the flowpath cross sectional areas was a laborious process that was, apparently, not particularly facilitated by the CAD system. Since the PIA geometry parameters are not particularly programmed for XY cross section generation only, it was a small matter for the PIA/LAPIN wrapper, upon determining that a two-dimensional inlet form was required, to request and obtain YZ cross sections that it then had processed and sorted to determine the correct cross-sectional area of the flowpath at each desired station.

PROCESS SPEED

It is tacitly assumed that anything computerized will, necessarily, happen in the blink of an eye. With regard to the acquisition, presentation, transfer, and consumption of geometric information reported above, this is not exactly the case. The PIA/CAD/CAPRI to PIA/LAPIN geometric information transfer took on the order of two days on a 1.5 GHtz, Pentium IV, workstation class machine. In contrast, the hand process that this technology could replace, took several weeks per case. This time reduction, while certainly a significant saving at one level, does not seem to match the expectations one might have of complete automation.

Two factors may serve to ameliorate this disappointment of expectation.

1. The considerable gain in reliability (that is, the elimination of mistakes that occur in a by-hand process) must be considered. It is not entirely appropriate that an iteration by automation be directly compared to an iteration by hand since a single iteration by hand has a much larger probability of being wrong. Perhaps it is more appropriate to compare a single iteration by automation to at least three iterations by hand so as to have at least a 2-out-of-3 confirmation that the right answer has been obtained by hand methods.

2. The present effort is a research effort only. The goal of the demonstration was to prove that the concepts of PIA did, in fact, work, not that they worked efficiently. In point of fact, many execution improvements could be implemented that would speed up execution in the event that the technology were to be applied as a day-to-day, working, production tool.

CONCLUSIONS

A wrapper conforming to the standards of the Project Integration Architecture has been developed. The wrapper provides a means for accessing geometry information from CAD files. Other PIA-conformant consumers of information (in the reported demonstration, another PIA-wrapped CFD analysis code, LAPIN) may then examine and consume that geometric information as appropriate. A demonstration of this transport of information has been successfully completed for the RBCC propulsion system.

To achieve a measure of CAD-vendor neutrality, the CAPRI geometric information access technology has been used. This allows the PIA/CAD wrapper to be easily reused when the supply of CAD information switches between vendors. Unfortunately, the present effort has not extended so far as to incorporate all CAPRI-supported CAD vendors in a single rendering of the PIA/CAD/CAPRI wrapper.

Finally, by using the reported technology, the need for transferring geometric information by hand from a CAD encapsulation to a consumer of that information, for example a CFD code, is eliminated. This hand process

can take several weeks for technologically advanced propulsion systems and is, regrettably, prone to undetected errors. By choosing the automated techniques enabled by the PIA technology, this process can be reduced to a matter of days (or less) and its reliability is distinctly increased.

The Project Integration Architecture provides a central system for accessing geometry data from CAD files and using that data for CFD simulations. To achieve the geometry data accessing functionality, CAPRI has been integrated into PIA. The benefits of wrapping CAPRI into PIA is that it offers direct access to data of many formats, provides accurate capture of information, and allows convenient data archiving in a single environment. All these benefits have been shown with this supersonic inlet simulation demonstration.

REFERENCES

1. Jones, William Henry. "Project Integration Architecture: Architectural Overview." NASA Glenn Research Center, 2001, PIA Web Site: http://www.grc.nasa.gov/WWW/price000/pub/tm07.pdf
2. Haimes, Robert. "Computational Analysis PRogramming Interface (CAPRI): A Solid Modeling Based Infrastructure for Engineering Analysis and Design." Cambridge, MA. November 1999. Web Site: http://raphael.mit.edu/capri/
3. Trefney, Charles. "An Air-Breathing Launch Vehicle Concept for Single-Stage-to-Orbit." NASA/TM—1999-209089, May 1999.
4. Cole, Gary and Richard, Jacques. "Supersonic Propulsion Simulation by Incorporating Component Models in the Large Perturbation Inlet (LAPIN) Computer Code." NASA TM–105193, December 1991.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2002	Technical Memorandum

4. TITLE AND SUBTITLE

Project Integration Architecture (PIA) and Computational Analysis Programming Interface (CAPRI) for Accessing Geometry Data From CAD Files

5. FUNDING NUMBERS

WU–704–01–13–00

6. AUTHOR(S)

Theresa L. Benyo

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, Ohio 44135–3191

8. PERFORMING ORGANIZATION REPORT NUMBER

E–13178

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546–0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA TM—2002-211358
AIAA–2002–0750

11. SUPPLEMENTARY NOTES

Prepared for the 40th Aerospace Sciences Meeting and Exhibit sponsored by the American Institute of Aeronautics and Astronautics, Reno, Nevada, January 14–17, 2002. Responsible person, Theresa L. Benyo, organization code 5880, 216–433–8723.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category: 61 Distribution: Nonstandard

Available electronically at http://gltrs.grc.nasa.gov/GLTRS

This publication is available from the NASA Center for AeroSpace Information, 301–621–0390.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Integration of a supersonic inlet simulation with a computer aided design (CAD) system is demonstrated. The integration is performed using the Project Integration Architecture (PIA). PIA provides a common environment for wrapping many types of applications. Accessing geometry data from CAD files is accomplished by incorporating appropriate function calls from the Computational Analysis Programming Interface (CAPRI). CAPRI is a CAD vendor neutral programming interface that aids in acquiring geometry data directly from CAD files. The benefits of wrapping a supersonic inlet simulation into PIA using CAPRI are; direct access of geometry data, accurate capture of geometry data, automatic conversion of data units, CAD vendor neutral operation, and on-line interactive history capture. This paper describes the PIA and the CAPRI wrapper and details the supersonic inlet simulation demonstration.

14. SUBJECT TERMS

Computer programs; Object-oriented programming; Systems integration; Supersonic inlets; Flow geometry; Computer aided design

15. NUMBER OF PAGES

16

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	